ADSS Server v8.4.0 – Go>Sign Developers Guide

ASCERTIA LTD

FEBRUARY 2026

Document Version- 1.0.0

CONTENTS

FIGURES

TABLES

1 Introduction

1.1 Scope

ADSS Go>Sign Service enables business applications to digitally sign documents using local or server held keys by use of Go>Sign Desktop application. In addition, it provides a powerful document viewer to display, navigate, and sign PDF documents. Go>Sign Service makes it easy to integrate Go>Sign Desktop into business applications. This document provides information on how to integrate ADSS Go>Sign Desktop in your business applications and take advantage of Go>Sign Service.

1.2 Intended Readership

This guide is intended for developers who are integrating business applications with ADSS Go>Sign Service. The document assumes a reasonable knowledge of web application development, specifically JavaScript and HTML.

1.3 Conventions

The following typographical conventions are used in this guide to help locate and identify information:

- **Bold** text identifies menu names, menu options, items you can click on the screen, file names, folder names, and keyboard keys.
- `Courier New` font identifies code and text that appears on the command line.
- **`Bold Courier New`** identifies commands that you are required to type in.

1.4 Technical support

If Technical Support is required, Ascertia has a dedicated support team providing debugging assistance, integration assistance and general customer support. Ascertia Support can be accessed in the following ways:

Website	https://www.ascertia.com
Email	support@ascertia.com
Knowledge Base	Ascertia Community Portal

In addition to the free support service describe above, Ascertia provides formal support agreements with all product sales. Please contact sales@ascertia.com for more details.

A Product Support Questionnaire should be completed to provide Ascertia Support with further information about your system environment. When requesting help, it is always important to confirm:

- System Platform details.
- ADSS Server version number and build date.
- Details of specific issue and the relevant steps taken to reproduce it.
- Database version and patch level.
- Product log files

2 ADSS Go>Sign Service Overview

ADSS Go>Sign Service empowers business applications to perform document signing on user's machines using the credentials held either locally by the user or on the server.

In addition, ADSS Go>Sign Service enables business applications to display PDF documents in a secure manner using a server-side HTML-based Go>Sign Viewer. It is fully integrated with Go>Sign PDF signing capabilities. This viewer allows users to view a document in flattened mode thus providing a "What You See Is What You Sign (WYSIWYS)" property.

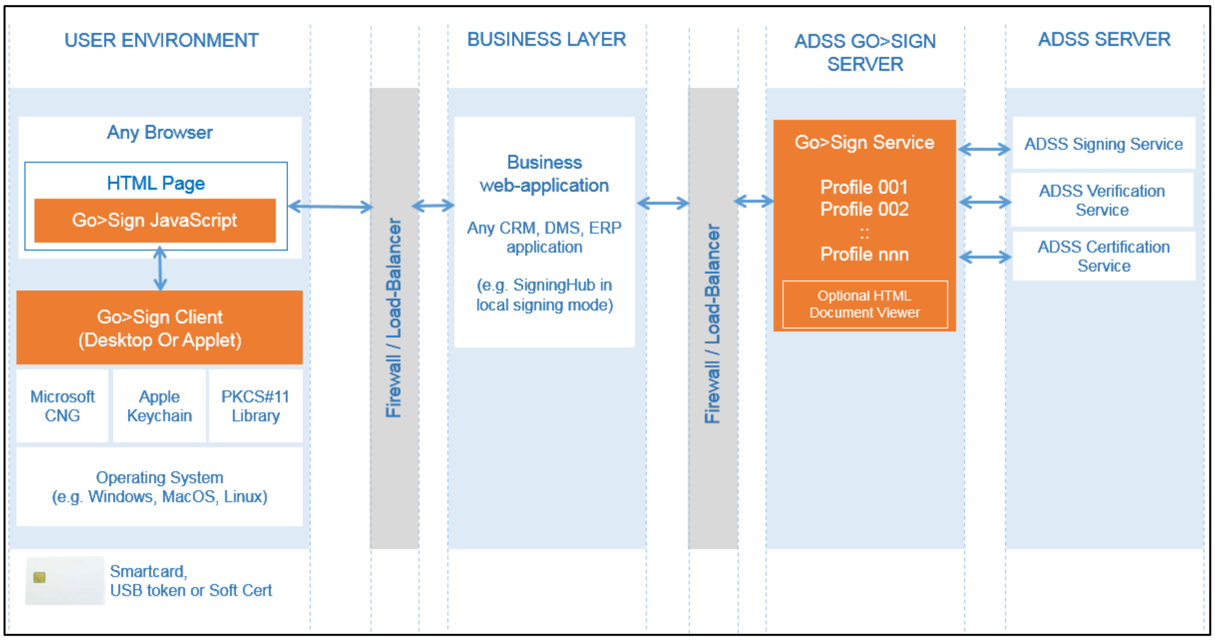


Figure 1 - ADSS Go>Sign Service & Business Application Interaction

The diagram shows how ADSS Go>Sign Service and business application interact with each other. The high-level process is as follows:

- An application web page sends a request to the ADSS Go>Sign Service specifying which profile it wishes to use, the data to be signed and other optional parameters.
- The ADSS Go>Sign Service receives the request and responds to web page with the relevant JavaScript code.
- The web page receives the JavaScript code and embeds it ready for the user.
- If the document is PDF or Word format and viewing has been requested, then the user can see the document and sign it using either locally-held or server-held signing key. If the document is another type or viewing was not requested, then the application must take responsibility for informing the user what it is that they are about to sign and Go>Sign asks the user to sign.
- As part of the signing process, ADSS Go>Sign Service can use the backend ADSS Server Services, to perform various tasks including creating server-side signatures, verify local signatures created by the user and to enhance basic signatures into long-term signature formats. Furthermore, if Go>Sign Service is used for key generation and certification, then ADSS Server can issue the certificates and securely store the user's private key.

2.1 Go>Sign Client Apps

The Go>Sign Client Apps have been designed to enable busy, non-technical people to easily and quickly sign documents and data using client held signing keys. It works with modern web browsers to allow citizens and businesses to go green by eliminating paper-based approvals, and thereby avoiding postage/courier, handling, storage and shredding costs.

The Go>Sign Client Apps are capable of creating signatures using locally-held signing keys (e.g. on a smartcard / secure USB token via PKCS#11 or software managed keys accessed through Windows CAPI or Mac Keychain keystore). As an advanced option Go>Sign Client Apps can generate keys pairs and certificate signing requests (PKCS#10), which can be certified by ADSS Server.

The Go>Sign Client Apps can sign documents using keys and certificates created on and stored by ADSS Server in PFX (PKCS#12) format. The user needs to provide the correct PFX password to sign a document or data object. Business applications can locally authenticate users e.g. using multi-factor authentication, before requesting ADSS Server to return their respective roaming credentials. One of the Go>Sign Client App available is Go>Sign Desktop.

2.1.1 Go>Sign Desktop

Go>Sign Desktop is a middleware application that allows users to sign the documents using locally held signing keys without using signed Java Applets. Browser vendors are discontinuing support for Java Applets, as Google Chrome has done already and thus Go>Sign Desktop provides a good alternative solution. Go>Sign Desktop is a small utility application that runs on the user's desktop and all communication is via JavaScript within the web browser session.

2.1.2 Go>Sign Client Apps Benefits

The main benefits of Go>Sign Client Apps are:

- They work as part of a web-browser environment and the associated web pages can be easily updated and functionality immediately rolled-out.
- They tightly control user interactions with the document through simple user interface with all elements being controlled so the user can only perform the actions which are wanted.
- They enable digital certificate filtering to allow the business to control which certificates the user can choose to sign with.
- Support for central / remote signing to be used as an alternative to using local keys and certificates
- ADSS Server hashes the data and just the hash (typically SHA-256) is passed to the Go>Sign Client Apps. The Go>Sign Service manages signature formatting, calling an appropriate verification profile to check the signature and certificate status and creating the correct format of signature. Formats supported including PDF, MS Word, XML and PKCS#7 / CMS.
- Time-stamped, long-term enabled digital signatures including ETSI PAdES, XAdES, and CAdES profiles.
- For PDFs full support is provided for PDF CDS and AATL signatures. Furthermore, visible and invisible signatures are supported, new and existing signature fields can be signed and certify signing and permissions are all available as options.
- Support for roaming credentials, whereby signing keys are held in a secure container on ADSS Server and provided to the Go>Sign Client Apps at the time of signing.
- Support for hardware signature tablet devices for drawing hand signatures e.g. Wacom, Signotec etc.
- Supports a wide variety of HTML5 browsers and platforms. For more information, see the System Requirements section.

2.1.3 System Requirements

Go>Sign **Desktop** is supported on the following operating systems and web browsers:

- Windows 11, Windows 10, Windows 8 and 8.1, Windows 7, Mac OS X 10.4 Tiger and above
- Internet Explorer 9.0+, Google Chrome 26.0+, Firefox 20.0+, Safari 5.0+, Microsoft Edge 20.0 on Windows

2.1.4 Supported Key Stores

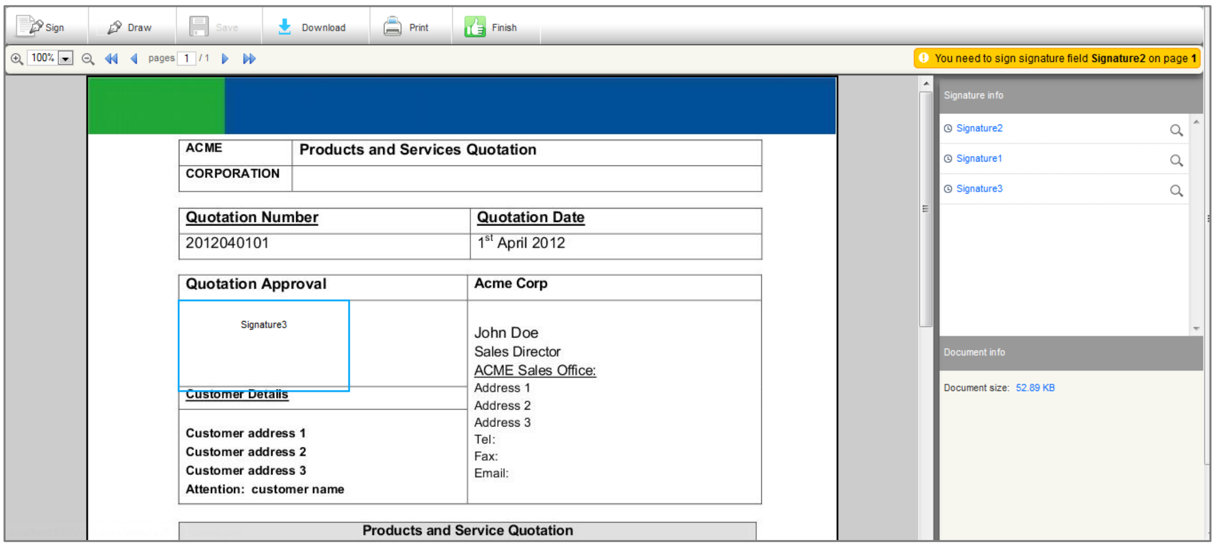
These cryptographic key stores, used to access the signing keys, are supported:

- MS CAPI/CNG (Windows)
- Mac Keychain keystore on Mac OS X and above
- PKCS#11 for hardware-based tokens
- Specific eID card implementations (these are separately licensed client specific keystore implementations e.g. the Emirates eID card, Belgian eID card via IntoIT middleware)
- Ascertia Roaming Keys

2.2 Go>Sign Viewer

The Go>Sign Viewer supports PDF and Word format documents and provides a number of features to aid the user experience such as page navigation, placing blank signature fields, signing operations, and document access right enforcement.

Go>Sign Viewer has an advanced viewer which images the document in stages a few pages at a time as needed. This allows large documents to be quickly opened and viewing started rather than waiting for the whole document to be imaged.



You need to sign signature field **Signature2** on page **1**

Document size: 52.89 KB

ACME	Products and Services Quotation
CORPORATION	

Quotation Number	Quotation Date
2012040101	1st April 2012

Quotation Approval	Acme Corp
Signature3	John Doe Sales Director ACME Sales Office: Address 1 Address 2 Address 3 Tel: Fax: Email:
Customer Details Customer address 1 Customer address 2 Customer address 3 Attention: customer name	

Products and Service Quotation

Figure 2 - Go>Sign Viewer Example

2.2.1 Go>Sign Viewer Benefits

The key features of Go>Sign Viewer are:

- Users can view and navigate both PDF and MS Word documents that are provided from the server or the user's local desktop, the document images are shown within the user's browser.

- Data leakage protection is provided with policy controlled download and print options as defined within the Go>Sign Service profile.
- The user can be allowed to create one or more blank signature fields in a PDF document.
- Existing blank signature fields and signature lines in PDF and MS Office Word documents can be signed.
- The business application can associate a signature field for a user to sign, and thus can control how and where the user signs the document.
- Existing signatures can be verified by clicking the target signature field (the verification is performed by the ADSS Server Verification Service making Trusted CA management simple and transparent to the user).
- PDF form filling is supported.
- The GUI and user visible messages support localization.
- The business application can control which buttons are visible on the Go>Sign Viewer toolbar.

3 ADSS Go>Sign Service Integration

Business applications can integrate with ADSS Go>Sign Service using standard HTTP messages.

The business application sends an HTTP POST request to the Go>Sign Service, which contains specific request headers and optionally the document to be signed. If required, the document to be signed is placed in the HTTP POST request body. ADSS Go>Sign Service responds with standard HTML and JavaScript code. The business application now embeds this in its own web page where the user will be asked to sign.

This table shows the request headers that can be used in the HTTP POST request:

Header Name	Mandatory	Description
ORIGINATOR_ID	YES	The client ID, which must be registered in ADSS Server Client Manager, and authorised to use ADSS Go>Sign Service.
USER_ID	NO	ADSS Go>Sign Service uses this parameter as a key alias if using server-side keys managed by ADSS Server (i.e. for server-side signing). Keys generated through Key Manager can be used for server-side signing. When a roaming key is used for client-side signing ADSS Go>Sign Service uses this parameter to locate the roaming key container from ADSS Certification Service. In the case of client side signing (when OS native API or PKCS#11 devices are used) this parameter is optional. This parameter is also used to set the user ID for authorise remote signing at ADSS RAS Service.
KEY_PASSWORD	NO	ADSS Go>Sign Service uses this parameter as a key password for the server-side keys held by ADSS Server (in PKCS#12/PFX format). This parameter is mandatory when a roaming key is generated. Note for Key Manager generated keys the value of this parameter is “NO_PASSWORD” and is a mandatory parameter.
USER_NAME	NO	Used for “%Signed_By%” field in server-side and mobile signing (used in PDF signature appearance). For multilingual characters, data should be sent in base64 format.
DATA_TO_BE_DISPLAYED	NO	Display message that will be shown to the user on mobile device when remote authorisation is enabled in Go>Sign Service. For multilingual characters, data should be sent in base64 format.
PROFILE_ID	NO	Defines Go>Sign profile identifier to be used by the web application. It can either be Profile ID or Profile Name
TRANSACTION_ID	NO	Defines the transaction identifier. This parameter is used in conjunction with parameter “REUSE_GOSIGN_SESSION” to save and recall the state of the document at Go>Sign Service. If the business application needs to perform multiple signing operations without uploading the document multiple times, it is a mandatory parameter.
REUSE_GOSIGN_	NO	Indicates that the document state at Go>Sign Service

SESSION		must be maintained if multiple Go>Sign profiles are utilized. Possible values of this parameter are 'true' and 'false'. If 'true', then document state is preserved and otherwise not. If the business application needs to perform multiple signing operations without uploading the document multiple times, it is a mandatory parameter.
FIELD_COORDS	NO	Used to create empty signature field(s) in the PDF document and signature appearance used at signing time. The value of this parameter is a comma-separated sequence representing X1, Y1, X2, Y2, page#, empty signature field name and signature appearance ID, e.g. 10,10,100,200,1,Signature1,appearance_id Multiple values are separated by "&" characters, e.g. 10,10,100,200,1,Signature1,appearance_id1&100,300,600,700,1,Signature2,appearance_id2 Note the provided signature appearance IDs must exist in the ADSS PDF Signature Appearances.
FIELD_NAME	NO	Represent an empty signature field name/signature line, and its value overrides the equivalent field name/signature line, as configured in Go>Sign profile. If set this will prevent the user from signing in any other field. If the business application wants the user to only sign the specific field, then this parameter is mandatory. Should the business application want the user to sign multiple fields then comma-separated field names are set in this parameter, e.g. Signature1, Signature2, Signature3. Multiple field signing is only supported for PDF documents using Go>Sign Viewer. For MS Office Signing both the suggested signer email address and setup ID for the signature line are supported.
SIGNATURE_APPEARANCE	NO	Signature appearance ID for PDF document signing. e.g. appearance_id. Note the provided signature appearance ID must exist in the ADSS PDF Signature Appearances.
DOCUMENT_ID	NO	Defines the document identifier provided by the business application. This is shown to the user in Go>Sign Viewer. Created and used by the business application for document management purposes.
DOCUMENT_NAME	NO	Name of the document that is displayed to the user in Go>Sign Viewer. This parameter is mandatory for MS Office signing. If Document Conversion feature is enabled in Go>Sign profile, then this parameter is mandatory. The value of this parameter must be the name of the document including its extension, e.g. contract.docx. The following file formats can be converted by Go>Sign Service: .doc, .docx, .xls, .xlsx, .ppt, .pptx, odt, sxw, .rtf, .txt, .ods, .csv, tsv and .tif. Created and used by the business application for document management purposes.
FILTER_SUBJECT_DN_	NO	String value upon which available certificates will be filtered on when the local key store is used for client side

CONTAINS		signing. Only those certificates whose subject DN value matches the provided string value will be listed. Possible values are: CN, OU, O, C e.g. CN=Test Certificate, O=Ascertia This will list all the certificates from the local key store that have a common name "Test Certificate" and organization "Ascertia" in the certificate's Subject DN.
FILTER_ISSUER_DN_CONTAINS	NO	String value upon which available certificates will be filtered on when the local key store is used for client side signing. Only those certificates whose issuer DN value matches the provided string value will be listed. Multiple issuer DN values can be specified using a "~" separated list. Possible values are: CN, OU, O, C e.g. CN=Test Issuer, O=Ascertia, CN= Test Issuer2, O=Ascertia This will list all the certificates from the local key store that have a common name "Test Issuer" and organization "Ascertia", or, that have a common name "Test Issuer2" and organization "Ascertia" in the certificate's issuer DN.
FILTER_SIGNATURE_ALGO_CONTAINS	NO	String value upon which available certificates will be filtered on when the local key store is used for client side signing. Only those certificates whose signature algorithm(s) value matches the provided string value will be listed. Multiple signature algorithms can be specified using a comma-separated list, e.g. SHA1WithRSA,SHA256WithRSA,SHA384WithRSA,SHA512WithRSA This will list all the certificates from the local key store whose certificate signature algorithm matches one of the stated values.
FILTER_POLICY_OID_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is sued for client side signing. Only those certificates whose certificate policy OID value matches the provided string value will be listed. Multiple certificate policy OIDs can be specified using a comma-separated list, e.g. 5.7.9.101.67.98.1.3 This will list all the certificates from the local key store whose certificate policy extension contains the stated policy OID(s).
FILTER_KU_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is sued for client side signing. Only those certificates whose certificate Key Usage value matches the provided string value will be listed. Multiple Key Usage definitions can be specified using a comma-separated list. Possible values are: digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly This will list all the certificates from the local key store whose certificate Key Usage extension contains one or more of those stated.

FILTER_EKU_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is sued for client side signing. Only those certificates whose certificate Extended Key Usage value matches the provided string value will be listed. Multiple Extended Key Usages definitions can be specified using a comma-separated list. The possible values are: clientAuth,emailProtection,smartCardLogon This will list all the certificates from the local key store whose certificate Extended Key Usage extension contains one or more of those stated.
FILTER_SHOW_EXPIRED_CERTIFICATES	NO	Boolean value to indicate if a filtered list of certificates should include expired certificates or not. Possible values are 'true' or 'false'.
FILTER_SHOW_QUALIFIED_CERTIFICATES	NO	Boolean value to indicate if a filtered list of certificates should include qualified certificates or not. Possible values are 'true' or 'false'.
FILTER_SAN_OTHER_NAME_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is used for client side signing. Only those certificates whose certificate otherName OID value matches the provided string value will be listed. Multiple certificate otherName OIDs can be specified using a comma-separated list, e.g. 5.7.9.101.67.98.1.3. This will list all the certificates from the local key store whose certificate Subject Alternative Name (SAN) extension contains the stated otherName OID(s).
FILTER_SAN_OTHER_NAME_VALUE_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is used for client side signing. Only those certificates whose certificate otherName value matches the provided string value will be listed. Multiple certificate otherName values can be specified using a '~&~' separated list, e.g. value1~&~value2. This will list all the certificates from the local key store whose certificate Subject Alternative Name (SAN) extension contains the stated otherName values.
FILTER_SAN_RFC822_NAME_CONTAINS	NO	Boolean value to indicate if a filtered list of certificates should include Subject Alternative Name (SAN) rfc822Name extension or not. Possible values are 'true' or 'false'.
FILTER_SAN_RFC822_NAME_VALUE_CONTAINS	NO	String value upon which available certificates will be filtered on when local key store is used for client side signing. Only those certificates whose certificate rfc822Name value matches the provided string value will be listed. Multiple certificate rfc822Name values can be specified using a '~&~' separated list, e.g. value1~&~value2. This will list all the certificates from the local key store whose certificate Subject Alternative Name (SAN) extension contains the stated rfc822Name values.
USER_LANGUAGE	NO	Defines the preferred language for Go>Sign Desktop and Go>Sign Viewer GUI and messages. Possible values are en, fr, de, etc. If this parameter is not specified, then

		default English language is used. If the business application wishes to display the messages in different languages based on user preferences, then this parameter is mandatory. To change the language preference, see Section 3.4.
REQUEST_TYPE	NO	Define the type of the request that is sent to Go>Sign Service. This parameter is used in conjunction with "TRANSACTION_ID" request parameter. When the business application wishes to send the hash of the document to Go>Sign Service instead of the whole document then this parameter is mandatory. Currently, the only accepted value for this parameter is "SET_HASH". When this parameter is used the business application must hash the document and send the resulting value in the body of the HTTP request instead of the actual document. Currently this feature is only supported for PDF documents
FINISH_URL	NO	Defines the URL hosted by the business application to which Go>Sign Viewer redirects the user to when they press the toolbar "Finish" button.
SIGNING_REASON	NO	Signing reason included in a signature.
SIGNING_LOCATION	NO	Signing location included in a signature.
SIGNER_CONTACT_INFORMATION	NO	Signer contact information included in a signature.
COMPANY_LOGO	NO	Defines whether or not to include the associated company logo in a visible PDF signature.
HAND_SIGNATURE_IMAGE	NO	Defines whether or not to include the hand signature image in a visible PDF signature.
CITY	NO	Defines the city signed attribute in XAdES signatures.
POSTAL_CODE	NO	Defines the postal code signed attribute in CAdES and XAdES signatures.
COUNTRY	NO	Defines the country signed attribute in CAdES and XAdES signatures.
STATE_OR_PROVINCE	NO	Defines the state or province signed attribute in CAdES and XAdES signatures.
COMMITMENT_TYPE_INDICATION	NO	Defines the commitment made by the signer. It is used as signed attribute in CAdES and XAdES signatures. It is used in PAdES signatures when explicit signature policy is set to ON in the signing profile.
SIGNER_ROLE	NO	Defines the signer role who generated the signature. It's used as a signed attribute in PAdES, CAdES and XAdES signatures.
DATA_OBJECT_FORMAT	NO	Defines the document format. It's used as a signed attribute in PAdES, CAdES and XAdES signatures.

SIGNING_ELEMENT_NAME	NO	This parameter defines the XML elements to be signed, these can a list of tag or element names or an XPath expressions, e.g. 'ContractName,ContractDate' or '//ContractName,//ContractDate'
USER_KEY	NO	This parameter uses to set the user signing key alias for remote authorise signing held at server (Software/HSM/Azure KeyVault etc.).
AUTH_TYPE	NO	This parameter defines the authentication type of registered user which is perform by ADSS RAS Service, e.g. BASIC_AUTH, NO_AUTH, SAML
AUTH_VALUE	NO	This parameter contains the base64 value of SAML assertion when AUTH_TYPE parameter is set to SAML.
SAN_EXTENSION	NO	String value to add Subject Alternative Name (SAN) extension during key/cert generation. Multiple SAN extensions can be specified using a '&' separated list. Possible values are: rfc822Name,otherName,iPAddress,dNSName, ediPartyName, uniformResourceIdentifer, registeredID e.g rfc822Name==value&dNSName==value&iPAddress==value&otherName==OID=value,encoding=UTF8String&ediPartyName==nameAssigner,encoding:UTF8String=partyName,encoding:IA5String&registeredID==1.2.3.4&uniformResourceIdentifier==https://ascertia.atlassian.net/browse/ADSS-8368 for multiple values use '~' seprator rfc822Name==value1~value2&dNSName==value1~value2&iPAddress==value1~value2&otherName==OID=value,encoding=UTF8String~OID=value,encoding=OctetString~OID=value,encoding=PrintableString&ediPartyName==nameAssignerValue,encoding:UTF8String=partyNameValue,encoding:IA5String&registeredID==1.2.3.4&uniformResourceIdentifier==https://ascertia.atlassian.net/browse/ADSS-8368.

Table 1 - ADSS Go>Sign Service HTTP Request Headers

When Go>Sign Viewer is used the business application does not need to use the Go>Sign Client Apps JavaScript methods and hence the integration is a simple process. However, if the business application does not use the Go>Sign Viewer it must utilise the JavaScript methods described in Section 4 to interact with the Go>Sign Client Apps.

3.1 Go>Sign Service Request (JSP Example)

Below is an example JSP web page code that details how the business application sends a request to ADSS Go>Sign Service. ADSS Go>Sign Service returns the HTML and JavaScript code snippet that is embedded within the web page and subsequently rendered by the web browser that allows the interaction with Go>Sign Client Apps.

```
<html>
<head>
```

```
<!-- Replace the localhost with the hostname/IP address where the ADSS
Go>Sign Service is running -->
        <script language="JavaScript"
src="http://localhost:8777/adss/gosign/applet/lib/adss_gosign.js"
type="text/javascript"></script>
        <script language="JavaScript"
src="http://localhost:8777/adss/gosign/script/jquery-1.9.1.min.js"
type="text/javascript"></script>
</head>
<body>
<%
    String str_docName = getServletContext().getRealPath("/") +
"/data/test_input_unsigned.pdf";
    FileInputStream fis = new FileInputStream(str_docName);
    byte[] m_byteArrDocument = new byte[fis.available()];
    fis.read(m_byteArrDocument);
    fis.close();
    URL obj_url = new URL("http://localhost:8777/adss/gosign/service");
    HttpURLConnection obj_http = (HttpURLConnection)
obj_url.openConnection();
    obj_http.setDoOutput(true);
```

```
    obj_http.setRequestMethod("POST");
    obj_http.setRequestProperty("Content-Length", m_byteArrDocument.length
+ "");
    obj_http.setRequestProperty("Content-Type", "application");
    obj_http.setRequestProperty("ORIGINATOR_ID", "samples_test_client");
    obj_http.setRequestProperty("PROFILE_ID", "adss:gosign:profile:001");
    OutputStream obj_out = obj_http.getOutputStream();
    obj_out.write(m_byteArrDocument);
    if (obj_http.getResponseCode() == 200) {
        BufferedReader obj_br = new BufferedReader(new
InputStreamReader(obj_http.getInputStream()));
        String str_line = "";
        while ((str_line = obj_br.readLine()) != null) {
            out.println(str_line);
        }
        obj_br.close();
    } else {
        out.println("HTTP Code : " + obj_http.getResponseCode());
    }
    obj_out.close();
%>
</body>
</html>
```

3.2 Go>Sign Service Request (ASP.NET / C# Example)

Below is an example ASP.NET (C#) code that details how the business application sends a request to ADSS Go>Sign Service. ADSS Go>Sign Service returns the HTML and JavaScript code snippet that is embedded within the web page and subsequently rendered by the web browser that allows the interaction with Go>Sign Client Apps.

```
String str_filePath = Server.MapPath("~/data/test_input_unsigned.pdf");
            byte[] documentbytes = File.ReadAllBytes(str_filePath);
            HttpWebRequest request= (HttpWebRequest)
HttpWebRequest.Create("http://localhost:8777/adss/gosign/service");
            request.Method = "POST";
            request.Headers.Add("ORIGINATOR_ID", "samples_test_client");
            request.ContentLength = documentbytes.Length;
            request.ContentType = "application";
            request.Headers.Add("PROFILE_ID", "adss:gosign:profile:001");
            Stream stream = request.GetRequestStream();
            stream.Write(documentbytes, 0, documentbytes.Length);
            HttpWebResponse httpresponse =
(HttpWebResponse)request.GetResponse();
            if (httpresponse != null)
            {
                StreamReader str = new
StreamReader(httpresponse.GetResponseStream());
                string viewer = str.ReadToEnd();
                GoSignViewer.InnerHtml = viewer;
            }
```

The associated web page should contain the required JavaScript imports to allow the received HTML and JavaScript code snippet to function correctly. For example:

```
<!-- Replace localhost with the hostname/IP address where the ADSS Go>Sign
Service is running -->
        <script language="JavaScript"
src="http://localhost:8777/adss/gosign/applet/lib/adss_gosign.js"
type="text/javascript"></script>
        <script language="JavaScript"
src="http://localhost:8777/adss/gosign/script/jquery-1.9.1.min.js"
type="text/javascript"></script>
```

3.3 How Business Applications Receive Signed Documents

The business application requests that ADSS Go>Sign Service returns the signed document. It sends an HTTP POST request with the necessary request headers. ADSS Go>Sign Service processes the request and responds back to the business application with an HTTP response containing the header information and the signed document. The signed document is placed in the body section of the HTTP response.

The following header is sent in the HTTP Post request from the business application to initiate the request:

Header Name	Mandatory	Description
TRANSACTION_ID	YES	Defines the transaction identifier for ADSS Go>Sign

		Service to process.

Table 2 - HTTP Request Headers to Initiate the Go>Sign Request

The following headers are sent in the HTTP response from the ADSS Go>Sign Service:

Header Name	Mandatory	Description
TRANSACTION_ID	YES	Defines the transaction identifier for the business application. Note this matches what was sent in the request.
USER_ID	NO	Defines the user identifier if that was provided by the business application in the initial ADSS Go>Sign Service request.
DOCUMENT_ID	NO	Defines the document identifier that was provided by the business application in the initial ADSS Go>Sign Service request.
SESSION_ID	NO	Defines the user session identifier if it was provided by the business application in the initial ADSS Go>Sign Service request. Note it may not have been sent in the initial request.
GOSIGN_RESPONSE_STATUS		Generic status identifier for the transaction. Possible values are: Success, Failure, Pending, and Declined.
MESSAGE		Message string as sent by the Go>Sign Service. Generally, this is only used when there is an error.

Table 3 - HTTP Response headers from Go>Sign Service

3.3.1 Signed Document Download Request (JSP Example)

Below is an example JSP web page code that details how the business application makes a request ADSS Go>Sign Service to retrieve a signed document:

```
<%
        URL obj_url = new URL("http://localhost:8777/adss/gosign/service");
        HttpURLConnection obj_http = (HttpURLConnection)
obj_url.openConnection();
        obj_http.setDoOutput(true);
        obj_http.setRequestMethod("POST");
        obj_http.setRequestProperty("Content-Type", "text/plain");
        // Gets the transaction identifier from the request parameter
        obj_http.setRequestProperty("TRANSACTION_ID",
request.getParameter("gosign_transaction_id"));
        byte[] m_byteArrDocument = null;
        if (obj_http.getResponseCode() == 200) {
            InputStream = obj_http.getInputStream();
            ByteArrayOutputStream buffer = new ByteArrayOutputStream();
            int nRead;
            byte[] data = new byte[1024];
            //reads the signed documents bytes from the HTTP response body.
            while ((nRead = inputStream.read(data, 0, data.length)) != -1)
{
                buffer.write(data, 0, nRead);
            }
            buffer.flush();
            m_byteArrDocument = buffer.toByteArray();
            buffer.close();
        } else {
            out.println("HTTP Code : " + obj_http.getResponseCode());
        }
%>
```

3.3.2 Signed Document Download Request (ASP.NET / C# Example)

Similarly, below is an example ASP.NET (C#) web page code that details how a business application makes a request to ADSS Go>Sign Service to retrieve the signed document:

```
            HttpWebRequest request =
(HttpWebRequest)HttpWebRequest.Create("http://localhost:8777/adss/gosign/se
rvice");
             request.Method = "POST";
             request.ContentType = "text/plain";
             request.Headers.Add("TRANSACTION_ID", str_id);
             HttpWebResponse httpresponse =
(HttpWebResponse)request.GetResponse();
             byte[] byteArr_doc = null;
             if (httpresponse != null)
             {
                 // reads the signed document bytes from the HTTP response
body.
    byteArr_doc = ReadBytesFromStream(httpresponse.GetResponseStream());
             }
```

3.4 Language Preference Settings

Go>Sign Desktop and Go>Sign Viewer both support localisation. The labels and messages shown by them can be displayed in different languages. To read more about how the business application can instruct Go>Sign Service to use different language, and for a list of supported configuration parameters, follow this link:

Language Manager

4 Go>Sign Client Apps JavaScript Library

If the business application is not using Go>Sign Viewer, then HTML and JavaScript code is required to connect with Go>Sign Desktop.

ADSS Go>Sign Service provides a JavaScript library (adss_gosign.js) that contains a number of JavaScript functions that are used to provide custom configurations for Go>Sign Client Apps, as well as to perform different actions using Go>Sign Client Apps.

4.1 Pre-processing Functions

- **GoSign_PostInit()**
  The business application must implement this JavaScript function in the same web page where the HTML and JavaScript code snippet received from Go>Sign Service is embedded.

  This method is automatically called implicitly once the Go>Sign Client Apps are initialized. In this method business applications can set additional configurations for Go>Sign Client Apps, e.g. set the HTML form name and HTML Select control name, i.e. to show a dropdown list by calling the **GoSign_SetFormName()** and **GoSign_SetCertificateListName()** respectively.

  This method can also be used by the business application to perform any custom functionality before the user signs a document, e.g. displaying instructions or a guidance message.

*The following functions must be called in the body of the **GoSign_PostInit()** function.*

- **GoSign_SetFormName(**formName**)**
  Set the name of the HTML form which contains the controls accessed by Go>Sign Client Apps.
- **GoSign_SetFileChooserName(**fileChooserName**)**
  Set the form field name for the HTML File control to read the file from the user machine.
- **GoSign_SetCertificateListName(**GoSignCertificateList**)**
  Set the form field name for the HTML List control which would be populated by Go>Sign Apps with the aliases of the certificates loaded from configured keystore.
- **GoSign_SetSigningReason(**signingReason**)**
  Set the signing reason attribute for a PDF signature.
- **GoSign_SetSigningLocation(**signingLocation**)**
  Set the signing location attribute for a PDF signature.
- **GoSign_SetContactInfo(**contactInfo**)**
  Sets the contact info attribute for a PDF signature.
- **GoSign_SetCertificateAlias(**certAlias**)**
  Set the alias of the certificate during roaming key registration. Alternatively, this value could be sent by Go>Sign Desktop to the Go>Sign processor for server side document signing.
- **GoSign_SetSubjectDN(**subjectDN**)**
  Set Subject DN for the certificate that will be generated during roaming key registration.
- **GoSign_SetAppletDialogColors(**titleColor,titleTextColor,backgroundColor,textColor,buttonsTextColor**)**
  Instruct Go>Sign Client Apps to use the provided colours in the password/PIN dialogs (PKCS#11, Roaming key, eID card etc.). The value of each colour must be in RGB format, e.g. title colour value should be specified as 200:200:200.
- **GoSign_SetAppletDialogTextFont(**fontName**)**
  Instruct Go>Sign Client Apps to display the text in the provided font in password/PIN dialogs

(PKCS#11, Roaming key, eID card etc.), e.g. Tahoma. For more information about the supported font names in Java, see the link below:

http://sanjaal.com/java/167/java-graphics/displaying-a-list-of-all-available-fonts-using-java/

- **GoSign_SetInputDocument(**inputDocToBeSigned**)**
  Used to directly pass the base64 encoded contents of the document to be signed to Go>Sign Client Apps.
- **GoSign_SetInputDocumentFieldName(**formFieldForInputDoc**)**
  Set the HTML form field name containing the base64 encoded contents of the input document to be signed by Go>Sign Client Apps.
- **GoSign_SetResultFilePostfix(**postfix**)**
  Set the result file postfix if the output document is to be stored locally on the user machine.
- **GoSign_SetOutputDocumentName(**outputDocumentName**)**
  Set the file name if the output document is to be stored locally on the user machine.
- **GoSign_SetOutputDocumentFieldName(**formFieldForSignedDoc**)**
  Set the HTML form field name where the base64 encoded contents of the signed document can be set by Go>Sign Client Apps.
- **GoSign_Base64Encode(**contentToBeEncoded**)**
  Utility function to base64 encode the textual or XML formatted contents.
- **GoSign_Base64Decode(**contentToBeDecoded**)**
  Utility function to decode the base64 encoded textual or XML formatted contents.
- **GoSign_ShowCertificates()**
  This function populates a drop down control with certificates fetched from a configured keystore based on specified filter criteria. The web application's drop down control should use an HTML Select control and the name of the field should be provided using the function GoSign_SetCertificateListName.

  NOTE: When using Go>Sign Desktop, a 'callback' function name must be passed as an argument to this function. The code snippet for the callback function should look like this:

```
GoSign_ShowCertificates(function(error){
        if(error != null){
            alert(error.errorCode +''+ error.message);
        }
});
```

- **GoSign_LoadCertificates()**
  This function loads the certificates in the background from the configured keystore. This is used when it is not required to show the certificates in a drop down list. If only one certificate is loaded, then it would automatically be used during the signing operation but if there are multiple certificates loaded then the first certificate from the list would be used.

  NOTE: When using Go>Sign Desktop, a 'callback' function name must be passed as an argument to this function. The code snippet for the callback function should look like this:

```
GoSign_LoadCertificates(function(error){
        if(error != null){
            alert(error.errorCode +''+ error.message);
        }
```

```
 });
```

- **GoSign_SetCallbackFunction(**callbackFunction**)**
  This method instructs Go>Sign Client Apps to invoke the business application call-back function once the signing operation is done. The call-back function will set an error object if an error occurred during the signing process. Error object contains an error code and message. If there is no error, then error object will be null.

  This method must be called before the **GoSign_Process()** function. The code snippet for the callback function looks like this:

```
function callbackFunction(error){
        if(error != null){
            alert(error.errorCode +''+ error.message);
        }
}
```

- **GoSign_GetVersion()**
  This method is used to get Go>Sign Desktop version. If client is using old version then this method will return '**NULL**' otherwise it will return proper build number of Go>Sign Desktop**.**

4.1.1 Qualified Certificate Generation in PKCS11 Physical Token

These methods provide enhanced cryptographic and certificate management features:

1. **GoSign_UseCCSlotForKeyGeneration**
   a. **Purpose**: Enables using CC slots for key generation.
   b. **Parameters**: option (Boolean - TRUE/FALSE).
   c. **Behavior**: When TRUE, GSD generates keys in the CC slot and disables Decrypt and Unwrap attributes.
   d. **Usage**: Applicable only when CryptoMode is set to PKCS#11.

2. **GoSign_DeleteOrphanKeysInPKCS11**
   a. **Purpose**: Deletes orphan keys (keys without an associated certificate) before importing a certificate.
   b. **Parameters**: option (Boolean - TRUE/FALSE).
   c. **Behavior**: When TRUE, orphan keys are deleted before the certificate import process.
   d. **Usage**: Applicable only when CryptoMode is set to PKCS#11.
   e. **Note**: Use cautiously, as it deletes keys permanently.

3. **GoSign_ValidateCertBeforeImportInPKCS11**
   a. **Purpose**: Validates end-entity certificates before importing to ensure an associated key exists.
   b. **Parameters**: option (Boolean - TRUE/FALSE).
   c. **Behavior**:
      i. Imports the certificate only if an associated key entry exists.
      ii. If it's a CA certificate, it is imported without validation.
      iii. Otherwise, an error is returned.
   d. **Usage**: Applicable only when CryptoMode is set to PKCS#11.

4. **GoSign_SetPkcs11PinParams**
   a. **Purpose:** Supply the PKCS#11/Token PIN in request so avoid displaying PIN/Password popup to user.
   b. **Parameters:**

i. **DeviceName:** Name of the PKCS#11 device.
ii. **UserPin:** Encrypted or Plain User PIN.

c. **Behavior:**
   i. Send an Encrypted or Plain Token PIN in request, if set Token PIN dialog will not be displayed to user.

d. **Usage:** Use this method when user interaction is not required to get Token PIN and displaying PIN dialog, In that case supply PIN in request.

4.1.2 Token Management

This section extends the PKCS#11 capabilities described in Section 4.1.1 by introducing advanced token management operations. These operations enable administrative control over the PKCS#11 physical token lifecycle, including token initialization, re-initialization, renaming, PIN recovery, and retrieval of enhanced token information. The methods described in this section are applicable only when CryptoMode is set to PKCS#11 and are intended to support Qualified Trust Service requirements and secure token administration scenarios:

1. **GoSign_InitNewPkcs11Token**
   a. **Purpose**: Initializes a brand-new or factory-reset PKCS#11 physical token by setting the Security Officer (SO) PIN, User PIN, and token label.
   b. **Parameters**:
      i. **DeviceName:** Name of the PKCS#11 device
      ii. **TokenID:** Unique identifier of the token
      iii. **DefaultSOPin:** New SO PIN (PUK)
      iv. **DefaultUserPIN:** New User PIN
      v. **TokenLabel:** Label to be assigned to the token
   c. **Behavior**:
      i. Initializes the token using the provided SO PIN and User PIN.
      ii. Creates a new token label.
      iii. Any existing data on the token is erased if the token was previously initialized.
   d. **Usage**: Used during first-time provisioning of PKCS#11 tokens or when onboarding new physical tokens.
   e. **Note:** This operation is destructive if the token was previously initialized.

The TokenID represents the slot identifier to which a specific token is attached. This value can be obtained only through the Token Details (TokenInfo) request, which returns the slot-related information for connected tokens. Once retrieved, the Business Application can use the TokenID to directly target and interact with the specific slot where the required token is connected.

Note: *If the TokenID is not provided and more than one token is connected to the system, a token selection dialog is displayed, allowing the user to choose the appropriate configured token.*

2. **GoSign_ReInitializePkcs11Token**
   a. **Purpose**: Completely resets an already-initialized PKCS#11 token by erasing all existing data and reapplying new credentials and label.
   b. **Parameters**:
      i. **DeviceName:** Name of the PKCS#11 device
      ii. **TokenID:** Unique identifier of the token
      iii. **OldSOPin:** Existing SO PIN
      iv. **DefaultSOPin:** New SO PIN
      v. **DefaultUserPIN:** New User PIN
      vi. **TokenLabel:** New token label
   c. **Behavior**:
      i. Authenticates using the existing SO PIN.

   ii. Erases all keys, certificates, and objects on the token.
   iii. Re-initializes the token with new SO PIN, User PIN, and label.

d. **Usage**: Applicable when tokens must be securely wiped and reissued due to policy, revocation, or operational changes.
e. **Note**: All cryptographic material on the token is permanently removed.

3. **GoSign_RenamePkcs11Token**
   a. **Purpose**: Renames an existing PKCS#11 token without modifying any stored data or PIN values.
   b. **Parameters**:
      i. **DeviceName:** Name of the PKCS#11 device
      ii. **TokenID:** Unique identifier of the token
      iii. **OldSOPin:** Existing SO PIN
      iv. **TokenLabel:** New token label
   c. **Behavior**:
      i. Authenticates using the SO PIN.
      ii. Updates only the token label.
      iii. Retains all keys, certificates, and PINs.
   d. **Usage**: Used when token labels must be updated for organizational, administrative, or identification purposes.

After a rename operation, the token may lose all existing data and configurations, which can require the User PIN to be initialized again. Because this operation can result in the complete removal of token content and settings, it is considered a critical action and should be performed with extreme caution.

4. **GoSign_ResetPkcs11UserPin**
   a. **Purpose:** Allows resetting the User PIN of a PKCS#11 token when the existing User PIN is known.
   b. **Parameters:**
      i. **DeviceName:** Name of the PKCS#11 device.
      ii. **PreviousPin:** Encrypted old User PIN.
      iii. **NewPin**: Encrypted new User PIN.
   c. **Behavior:**
      i. Sends a request to GoSign Desktop to update the User PIN.
      ii. The token validates the existing PIN before applying the new PIN.
   d. **Usage:** Used for standard User PIN change operations.

5. **GoSign_ResetPkcs11SoPin**
   a. **Purpose**: Allows resetting the Security Officer (SO) PIN for a PKCS#11 token.
   b. **Parameters**:
      i. **DeviceName:** Name of the PKCS#11 device.
      ii. **PreviousPin:** Encrypted old SO PIN.
      iii. **NewPin**: Encrypted new SO PIN.
   c. **Behavior**
      i. Authenticates using the existing SO PIN.
      ii. Replaces it with the new SO PIN.
   d. **Usage:** Use the encrypted values of PreviousPin and NewPin for secure transmission.

6. **GoSign_ResetPkcs11UserPinUsingSO**
   a. **Purpose:** Resets the User PIN using the SO PIN (PUK) when the User PIN is unavailable or forgotten.
   b. **Parameters**:
      i. **DeviceName:** Name of the PKCS#11 device
      ii. **TokenID:** Token identifier
      iii. **SoPIN:** SO PIN / PUK
      iv. **NewUserPin:** New User PIN.

c. **Behavior**:
   i. Authenticates to the token as Security Officer.
   ii. Forcefully initializes the User PIN using SO privileges.
d. **Usage:** Applicable in PIN recovery scenarios where the User PIN cannot be provided.
e. **Note:** This operation bypasses the need for the previous User PIN.

7. **GoSign_GetTokenInfo**
   a. **Purpose:** Retrieves detailed information about all connected and configured PKCS#11 tokens.
   b. **Parameters:** None
   c. **Behavior:**
      i. Returns token metadata including label, serial number, manufacturer, memory status, and initialization state.
      ii. Indicates whether the token and User PIN are initialized.
   d. **Usage:** Used to inspect token state, validate readiness, and support diagnostic or administrative workflows.

4.1.3 Securely Supply PKCS11 PIN/PUK in Request

1. A new Encrypt button has been introduced in the PKCS11 GoSign demo for Java, allowing users to input multiple PINs for encryption. The encryption process is performed using the Go>Sign Service.

   Users can also directly test the service using Postman. The sample cURL request is shown below:

```
 curl --location 'http://localhost:8777/adss/gosign/service' \
 --header 'Content-Type: application' \
 --header 'ORIGINATOR_ID: samples_test_client' \
 --header 'REQUEST_TYPE: ENCRYPT' \
 --data '{
   "data": [
     "string1",
     "string2",
     "string3"
   ]
}'
```

2. To send an encrypted PIN, submit the above request to the Go>Sign Service.
3. The encrypted PIN received in the response can be supply to Go>Sign Desktop (GSD) in following methods:
   a. GoSign_SetPkcs11PinParams
   b. GoSign_ResetPkcs11UserPin
   c. GoSign_ResetPkcs11SoPin
   d. GoSign_InitNewPkcs11Token
   e. GoSign_ReInitializePkcs11Token
   f. GoSign_RenamePkcs11Token
   g. GoSign_ResetPkcs11UserPinUsingSO
4. Go>Sign Desktop will attempt to decrypt the supplied PIN/PUK.

a. If decryption is successful, the decrypted value will be used as the PIN/PUK.

b. If decryption fails, the original supplied PIN/PUK will be used directly.

4.2 Processing Functions

- **GoSign_Process()**
  Starts the document signing process during which the user may be asked to provide a PIN or password to access the locally held signing key.

  Once signature processing is completed the configured callback function is automatically invoked. The callback function that is invoked should have been configured using the function **GoSign_SetCallbackFunction**.

4.3 Post Processing Functions

- **GoSign_GetTransactionId()**
  This method is only applicable when Go>Sign Client Apps are embedded in the web page. It returns the transaction identifier that will be used by the business application to retrieve the signed document from ADSS Go>Sign Service.
- **GoSign_GetErrorCode()**
  If the requested operation fails, then this method can be called to retrieve the error code. Each error code is mapped to a readable error message string from the appropriate messages language file.
- **GoSign_GetErrorReason()**
  If the requested operation fails, then this method can be called to retrieve the readable error message string from appropriate messages language file.
- **GoSign_GetRaTransactionId()**
  Return Transaction ID generated by a Registration Authority(RA) when Go>Sign Service sends a certificate signing request to RA. It is a mandatory request parameter in Go>Sign Service in order to retrieve the generated certificate.
- **GoSign_GetOutputDocument()**
  Return the base64 encoded contents of the signed documents.

5 Go>Sign Viewer JavaScript Functions

Go>Sign Viewer provides several JavaScript functions that facilitate the business application to retrieve information from the Go>Sign Viewer.

- **window.pdfviewer.getTransactionId()**
  Return the transaction identifier that will be used by the business application to get the signed document from ADSS Go>Sign Service, or used in communication with ADSS Go>Sign Service for other functionality.
- **GoSign_NotifyStateChange(status_code)**
  Sometimes business applications wish to perform some custom functionality as soon as an operation is performed by a user within Go>Sign Viewer. For example, Go>Sign Viewer notifies the host web page of the business application that a signature field has been created, signed or a form field has been saved. In such scenarios the business application can implement this function in the host webpage in order to get a notification update on the document state change. The possible values for a status_code argument passed by Go>Sign Viewer are:
  - **changed_incomplete**
    User didn't fill all the mandatory fields in the document)
  - **changed**
    The user created signature field(s) or filled the form field(s)
  - **signed**
    A signature field is signed by the user
  - **declined**
    User pressed the Decline button
  - **error**
    Some error occurred

  The code snippet for this function should look like this:

```
function GoSign_NotifyStateChange(status_code){
         if(status_code == ”changed_incomplete”){
            alert(“All mandatory fields in the PDF form are not
 filled”);
         }else if(status_code == ”changed”){
            alert(“Signature field(s) created or/and form field(s)
 saved”);
         }else if(status_code == “signed”){
                   alert(“A signature field is signed by user”);
         } else if(status_code == “error”){
                   alert(“Some error occurred”);
      }
 }
```

Clicking the Finish button in Go>Sign Viewer toolbar redirects the browser to the business application webpage address configured in Go>Sign Service profile or passed as FINISH_URL parameter to Go>Sign Service. The business application webpage receives the following query parameters from Go>Sign Viewer, which it can use to continue further processing at business application end:

- **gosign_transaction_id**

The value of this parameter is a unique transaction Id generated by Go>Sign Service for this transaction which can be used to download the updated or signed document from Go>Sign Service.

- **gosign_status**
  The value of this parameter provides the status of the last operation performed by user in Go>Sign Viewer. The possible values of this parameter are:
  - **unchanged**
    The user made no changes or actions document.
  - **changed_incomplete**
    The user didn't fill all the mandatory fields in the document.
  - **changed**
    The user created signature field(s) or filled the form field(s).
  - **signed**
    A signature field is signed by the user.
  - **declined**
    User pressed the Decline button.
  - **gosign_message**
    If user declined the transaction and provided a declining reason then this parameter provides the declining reason to the business application.

6 Example Scenarios & Demos for Go>Sign Service

ADSS Client SDK (Java and .NET) ships with a number of web based Go>Sign Service demos, which highlight how ADSS Go>Sign Service can be used in different business scenarios. These demos show how ADSS Go>Sign Service profiles provide a powerful platform for business applications to easily implement complex signing processes.

For information on how to configure the different ADSS Go>Sign Service Profiles, see the ADSS Server Admin Manual:

Step 1 - Creating a Go-Sign Profile

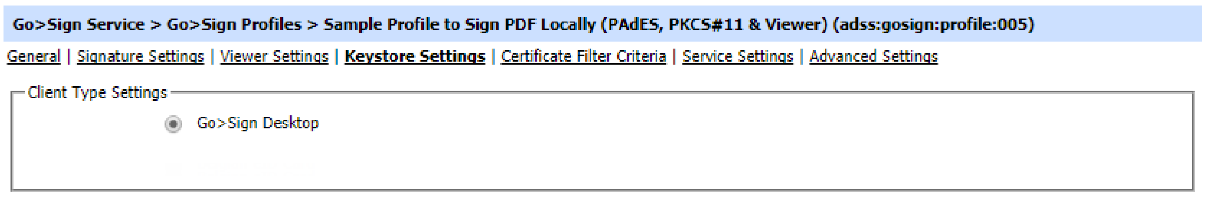
- To run Go>Sign demo application without any changes, ADSS Server, Go>Sign Desktop and ADSS Client SDK (Java/.NET) must be installed on the same machine. In addition, the web browser should be accessed from the same machine where both ADSS Server and ADSS Client SDK (Java/.NET) are installed. To install Go>Sign Desktop see the **ADSS-Go-Sign-Desktop-Installation-Guide.pdf** at location

  <ADSS Client SDK Directory>/GoSign/Desktop/docs.
- If ADSS Server and ADSS Client SDK (Java/.NET) are installed on different machines, then
  - Go>Sign demo pages must be modified to specify the address of the ADSS Server machine
  - Go>Sign Service Address configuration must be updated using ADSS Server Console to reflect the IP address or host name. Either at **Service Manager** screen or inside the required **Go>Sign Profile** under **advanced settings** tab.

Once Go>Sign demo web application has been deployed successfully, the user can access each demo by launching the **available_demos.html** file available at location:

<ADSS Client SDK>/GoSign/Demo

All the Go>Sign demos can be tested with desktop client by setting the appropriate configuration item in the respective Go>Sign profile in the ADSS Server Console as shown in the figure.

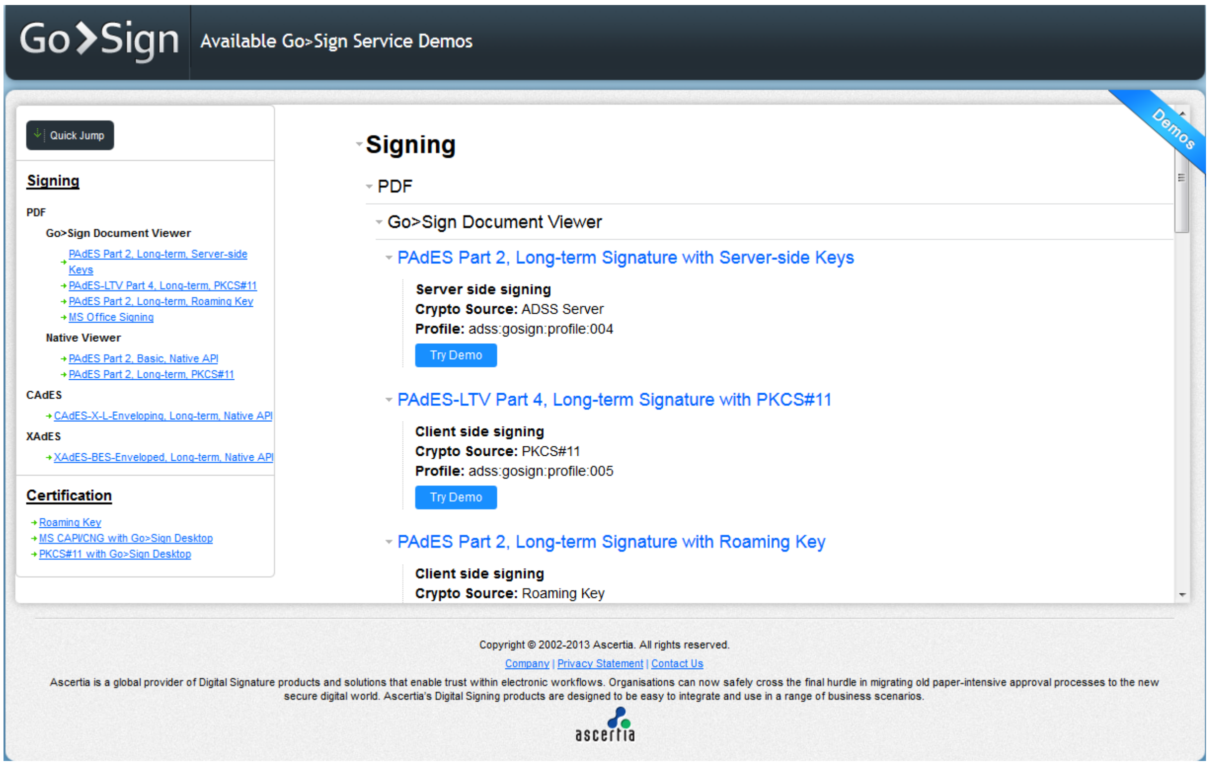


Go>Sign Service > Go>Sign Profiles > Sample Profile to Sign PDF Locally (PAdES, PKCS#11 & Viewer) (adss:gosign:profile:005)

General | Signature Settings | Viewer Settings | **Keystore Settings** | Certificate Filter Criteria | Service Settings | Advanced Settings

Client Type Settings

● Go>Sign Desktop



Signing

PDF

Go>Sign Document Viewer

- PAdES Part 2, Long-term, Server-side Keys
- PAdES-LTV Part 4, Long-term, PKCS#11
- PAdES Part 2, Long-term, Roaming Key
- MS Office Signing

Native Viewer

- PAdES Part 2, Basic, Native API
- PAdES Part 2, Long-term, PKCS#11

CAdES

- CAdES-X-L-Enveloping, Long-term, Native API

XAdES

- XAdES-BES-Enveloped, Long-term, Native API

Certification

- Roaming Key
- MS CAPI/CNG with Go>Sign Desktop
- PKCS#11 with Go>Sign Desktop

Signing

PDF

Go>Sign Document Viewer

PAdES Part 2, Long-term Signature with Server-side Keys

Server side signing
Crypto Source: ADSS Server
Profile: adss:gosign:profile:004

PAdES-LTV Part 4, Long-term Signature with PKCS#11

Client side signing
Crypto Source: PKCS#11
Profile: adss:gosign:profile:005

PAdES Part 2, Long-term Signature with Roaming Key

Client side signing
Crypto Source: Roaming Key

Copyright © 2002-2013 Ascertia. All rights reserved.

Company | Privacy Statement | Contact Us

Ascertia is a global provider of Digital Signature products and solutions that enable trust within electronic workflows. Organisations can now safely cross the final hurdle in migrating old paper-intensive approval processes to the new secure digital world. Ascertia's Digital Signing products are designed to be easy to integrate and use in a range of business scenarios.

Figure 3 - Go>Sign Demo Parking Page

The sections below explain each demo individually in detail and provide information about how the demo works, which configurations are applied in Go>Sign profile, and where the source files of a particular demo are available.

6.1 Document Signing Demos

6.1.1 Go>Sign Viewer – PAdES Part 2, Long-term Signature with Server-side Keys

This is another common usage mode, which uses Go>Sign Viewer to view the document. The signing of the document is performed by ADSS Server. Go>Sign Viewer is only used to display the PDF document, create blank signature fields and sign aforementioned fields using server side keys held by ADSS Server.

This demo uses the sample Go>Sign profile (adss:gosign:profile:004) configured in the ADSS Go>Sign Service.

The user can access this demo using the following URL(s):

- http://localhost:8766/pdf-sig-viewer-remote (Java)
- http://localhost/pdf-sig-viewer-remote (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/PDF-Sig-Viewer-Remote

6.1.2 Go>Sign Viewer – PAdES-LTV Part 4, Long-term Signature with PKCS#11

Another common usage mode, which uses Go>Sign Viewer to display PDF documents, create blank signature fields and sign the document. ADSS Go>Sign Service computes the hash of the document, Go>Sign Desktop performs hash signing and ADSS Go>Sign Service verifies and enhances the PAdES signature into a Long-Term Verification format and assembles the final PDF.

This demo uses the PKCS# 11 keystore, and the sample Go>Sign profile (adss:gosign:profile:005) configured in the ADSS Go>Sign Service. By default, the PKCS#11 device (Aladdin) is configured in the Go>Sign profile.

Pre-requisites:

In order to run this demo, make sure that your desired PKCS#11 device is configured in Go>Sign profile. Ensure that you restart the Go>Sign Service after any updates.

The user can access this demo using the following URL(s):

- http://localhost:8766/pades-sig-viewer-local (Java)
- http://localhost/pades-sig-viewer-local (.Net)

The source code for web pages used in this demo can be found in ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/Pades-Sig-Viewer-Local

6.1.3 Go>Sign Viewer – PAdES Part 2, Long-term Signature with Roaming Key

This is another fairly common usage mode using the power of Go>Sign Desktop with its Go>Sign Viewer to control the display of a PDF. The Go>Sign Viewer is used to display a PDF document, creating blank signatory fields and subsequent signing. ADSS Go>Sign Service handles the document hash computation, Go>Sign Desktop performs hash signing, ADSS Go>Sign Service uses ADSS Server to verify and enhance the signature into an advanced format, and finally, the enhanced signature is assembled into the PDF document by the ADSS Go>Sign Service.

This demo uses a roaming keystore, and the sample Go>Sign profile (adss:gosign:profile:003) configured in the ADSS Go>Sign Service. A roaming key is used for local signing (of course it can easily be configured to use locally held CAP/PKCS#11 based key or server-side key).

Pre-requisites**:**

In order to run this demo, first run the demo defined in Section 6.2.1 with key alias value “ROAMING_KEY_ALIAS”.

The user can access this demo using the following URL(s):

- http://localhost:8766/pdf-sig-viewer (Java)
- http://localhost/pdf-sig-viewer (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/PDF-Sig-Viewer

6.1.4 Native Viewer – PAdES Part 2, Basic Signature with Native API

This is a common usage scenario of Go>Sign Desktop without Go>Sign Viewer. ADSS Go>Sign Service is responsible for producing the document hash. Go>Sign Desktop signs the hash and finally, ADSS Go>Sign Service assembles the signature into the original document. If the business application displays the PDF to the user, then Adobe Reader (or any installed PDF Reader) is used and this can validate the signature locally using whatever local trust anchors exist on the user’s system.

This demo uses the OS native API (MSCAPI & Mac Keychain). It uses the sample Go>Sign profile (adss:gosign:profile:001) configured in ADSS Go>Sign Service.

The user can access this demo using the following URL(s):

- http://localhost:8766/pdf-sig (Java)
- http://localhost/pdf-sig (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/PDF-Sig

6.1.5 Native Viewer – PAdES Part 2, Long-term Signature with PKCS#11

This is a common usage scenario of Go>Sign Desktop when it is used with a native document viewer, e.g. Adobe Reader. Go>Sign Service handles the document hash computation, Go>Sign Desktop signs the computed hash, and the Go>Sign Service assembles and enhances the signature to Long Term Verification format with the help of ADSS Server.

This demo uses PKSC#11 keystore, and the sample Go>Sign profile (adss:gosign:profile:002) configured in the ADSS Go>Sign Service. By default, the PKCS#11 device (Aladdin) is configured in the Go>Sign profile.

Pre-requisites:

In order to run this demo, make sure that your desired PKCS#11 device is configured in Go>Sign profile. Ensure that you restart the Go>Sign Service after any updates.

The user can access this demo using the following URL(s):

- http://localhost:8766/pdf-sig-local (Java)
- http://localhost/pdf-sig-local (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/PDF-Sig-Local

6.1.6 CAdES-X-L-Enveloping, Long-term Signature with Native API

This demo shows how CAdES-BES & CAdES-XL signatures can be computed over any type of document using Go>Sign Desktop.

This demo uses the OS native API (MSCAPI & Mac Keychain), and the sample Go>Sign profile (adss:gosign:profile:008) configured in the ADSS Go>Sign Service.

The user can access this demo using the following URL(s):

- http://localhost:8766/file-sig (Java)
- http://localhost/file-sig (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/File-Sig

- file_signature.jsp
- result.jsp

6.1.7 CAdES-B-B-Enveloping, with Roaming Key

This demo illustrates the computation of CAdES-BES and CAdES-B-B signatures on any document type using Go>Sign Desktop. It utilizes the roaming key and the predefined Go>Sign profile (adss:gosign:profile:008) set up in the ADSS Go>Sign Service. Make sure to select the Roaming key option in the key store settings of this profile before running the demo.

The user can access this demo using the following URL(s):

- http://localhost:8766/file-sig/File_Signature_Roaming?userid=RoamingCert (Java)
- http://localhost/file-sig/File_Signature_Roaming?userid=RoamingCert (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/File-Sig

- file_signature_roaming.jsp
- result_roaming.jsp

6.1.8 XAdES-BES-Enveloped, Long-term Signature with Native API

This is a common usage scenario of Go>Sign Desktop for XML signatures.

This demo uses the OS native API (MSCAPI & Mac Keychain), and the sample Go>Sign profile (adss:gosign:profile:006) configured in the ADSS Go>Sign Service.

The user can access this demo using the following URL(s):

- http://localhost:8766/xml-sig (Java)
- http://localhost/xml-sig (.Net)

The source code for web pages used in this demo can be found in ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/XML-Sig

6.1.9 e-Tendering (XAdES-BES-Enveloped, Long-term Signature with Native API)

This demo is not available by default in ADSS Client SDK. If you want to see how e-Tendering works, then you should request the special ADSS Server license and a demo package by sending an email to support@ascertia.com.

This is a special usage scenario in which Go>Sign Desktop is used to create an XML signature.

Description:

1. A tender is created on the server by demo web application.
2. Web application creates a certificate for tender using Certification Service.
3. User selects an already created tender for which he wants to submit a response.
4. User is presented with a web form where he can browse a document from his local machine. The form contains a drop down list which is populated with a number of certificates loaded from the configured keystore.
5. User browses and selects a document from his local machine.
6. User selects a certificate from the drop down list.
7. User clicks Secure Upload button to proceed with signing and encryption of the document.
8. XML document is signed by Go>Sign Desktop and then encrypted by Go>Sign Service using the tender certificate and the final document is retrieved by the web application and a response entry is shown in the web application against the tender.
9. User clicks to view the available tender responses and then he is presented with a list of such responses.

10. User clicks Decrypt to decrypt the tender response, and the tender is decrypted using Decryption Service and three possible options are displayed to user: verify the response signature (using Verification Service), download the PayLoad (original user document) or download the signed tender response XML.
11. User can choose either of the provided option to see the results.

This demo uses the OS native API (MSCAPI & Mac Keychain), and the sample Go>Sign profile (adss:gosign:profile:006) configured in the ADSS Go>Sign Service.

Pre-requisites:

In order to run this demo, please ensure the following:

- ENCRYPTION must be enabled for Go>Sign Service in ADSS Server license.
- Update Go>Sign profile (adss:gosign:profile:006) and apply the following changes:
    - In General screen, change the Document Input Source to “Client”.
    - In Signature Settings screen check the “Encrypt XML after Signing” checkbox and provide the XML element name to be encrypted.
    - In Service Settings screen provide the Certificate Service Settings.
    - Decryption Service must be enabled in ADSS Server license.

The user can access this demo using the following URL(s):

- http://localhost:8766/e-tendering (Java)
- http://localhost/e-tendering (.Net)

The source code for web pages used in this demo can be found in ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/E-Tendering

6.2 Certification Demos

6.2.1 Certificate Generation – Roaming Key with Go>Sign Desktop

This demo shows how Go>Sign Desktop generates a roaming key pair and how ADSS Server generates a roaming certificate for this key pair. ADSS Server stores this roaming key credential which can later be used by Go>Sign Desktop for signing documents.

This demo uses the sample Go>Sign profile (adss:gosign:profile:009) configured in the ADSS Go>Sign Service.

- Ensure ADSS Go>Sign Desktop is installed on client machines.
- In the following code files, rename localhost with the IP Address of the ADSS Server machine if your code is running on a separate machine than the ADSS Server machine:

  (Roaming_key_certificate.aspx, Roaming_key_certificate.aspx.cs) or (roaming_key_certificate.jsp, roaming_key_certificate_retrieve.jsp)

The user can access this demo using the following URL(s):

- http://localhost:8766/roaming (Java)
- http://localhost/roaming (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/tomcat/webapps/Roaming

6.2.2 Certificate Generation – MS CAPI/CNG with Go>Sign Desktop

This demo shows how Go>Sign Desktop locally generates a key pair and how ADSS Server generates a certificate for this key pair. Go>Sign Desktop stores this key and the certificate in the MSCAPI key store, which can be used by Go>Sign Desktop for signing documents.

This demo uses the sample Go>Sign profile (adss:gosign:profile:010) configured in the ADSS Go>Sign Service.

- Ensure ADSS Go>Sign Desktop is installed on client machines
- In the following code files, rename localhost with the IP Address of the ADSS Server machine if your code is running on a separate machine than the ADSS Server machine: (Mscapi_key_certificate.aspx,
  Mscapi_key_certificate_retrieve.aspx,
  Mscapi_key_certificate.aspx.cs,
  Mscapi_key_certificate_retrieve.aspx.cs)
  or (mscapi_key_certificate.jsp, mscapi_key_certificate_retrieve.jsp)

The user can access this demo using the following URL(s):

- http://localhost:8766/mscapi (Java)
- http://localhost/mscapi (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/MSCAPI

6.2.3 Certificate Generation – PKCS#11 with Go>Sign Desktop

This demo shows how Go>Sign Desktop locally generates a key pair and how ADSS RA Service, along with Certification Service, generates a certificate for this key pair. Go>Sign Desktop stores this key and the certificate in the PKCS#11 device, which can later be used by Go>Sign Desktop for signing documents.

This demo uses the sample Go>Sign profile (adss:gosign:profile:010) configured in the ADSS Go>Sign Service.

Pre-requisites:

In order to run this demo, please ensure the following:

- Ensure ADSS Go>Sign Desktop is installed on client machines.
- ADSS RA (Registration Service) must be enabled in ADSS Server license.
- To issue the certificate right away, in the RA Profile adss:ra:profile:001 enable 'Allow auto-approval for web based requests (no manual approval required)' otherwise the RA Operator first needs to approve the pending request to later import it.
- Update Go>Sign profile (adss:gosign:profile:010) and apply the following changes:
    - In Key Store Settings screen change the Key store settings to PKCS#11 and provide PKCS#11 configuration settings.
    - In Service Settings screen select the “Use RA Service to generate certificate” option and provide the RA Service address along with RA profile.
- In the following code files, rename localhost with the IP Address of the ADSS Server machine if your code is running on a separate machine than the ADSS Server machine: (Pkcs11_key_certificate.aspx,
  Pkcs11_key_certificate_retrieve.aspx,
  Pkcs11_key_certificate.aspx.cs,
  Pkcs11_key_certificate_retrieve.aspx.cs)
  or (pkcs11_key_certificate.jsp, pkcs11_key_certificate_retrieve.jsp)

The user can access this demo using the following URL(s):

- http://localhost:8766/pkcs11-ra (Java)
- http://localhost/pkcs11-ra (.Net)

The source code for web pages used in this demo can be found in the ADSS Client SDK under the following path:

<ClientSDK>/GoSign/Demo/GoSignDemos/PKCS11-RA

7 Java Go>Sign Demo – Deployment & Configuration

7.1 Deploying the Java Demo Application

The Go>Sign Desktop demo web application is written using Java web technologies. The demo application is shipped with Tomcat Application Server v10.1.42.

To deploy the application, extract the supplied ZIP file into a directory, e.g.
GoSignDemoInstallation.

- Navigate to the location **<GoSignDemoInstallation>\GoSign\Demo\tomcat\bin\**
- Edit the file **setclasspath.bat** file in a text editor, set the JDK path in JAVA_HOME variable e.g.

  SET JAVA_HOME= C:/Program Files/Java//jdk17.0.18
- Execute the **startup.bat** file to run the Tomcat. You can shut down the Tomcat server by executing the **shutdown.bat** file.

By default, the port used by Tomcat is 8766. If it is required to change the default port then follow these instructions:

- *Navigate to location:* ***<GoSignDemoInstallation>\GoSign\Demo\tomcat\conf***
- *Edit the* ***server.xml*** *in a text editor*
- *Search for the parameter* ***<Connector port="8766"*** *and change the port accordingly*
- *Restart the tomcat to have the changes take effect.*

8 NET Go>Sign Demo – Deployment & Configuration

8.1 Deploying the .Net Demo Application on IIS 7.5 or higher (v4.5)

The steps to deploy .NET Go>Sign Applet Demo on IIS 7.5 or higher are as follows:

1. Open IIS Manager.
2. Open **Sites** > **Default Web Site** from the navigation tree in the left-hand pane.
3. Right-click on the **Default Web Site** tree node and select **Add Application**. A wizard will be launched.
4. Enter the name of the web application in the 'Alias' field, e.g. pdf-sig and select the path to the web application, i.e. **<GoSignDemoInstallation>\GoSign\Demo\GoSignDemos\Pdf-Sig**.
5. Right-click on the pdf-sig web application and then edit the permissions giving full access rights to authorised users.
6. Right-click on the pdf-sig application again and select **Manage Application, Browse**. This will open your default web browser and you will see the main page of the .NET Go>Sign Demo. Select one of the options to run the demo. (Alternatively, open your browser and enter http://localhost/pdf-sig as the URL. You will see the main page for the .Net Go>Sign Demo).

8.2 Deploying the .Net Demo Application on IIS 7.5 or higher (v8.0)

The .NET Core 8.0 Go>Sign Applet Demo can also be deployed on IIS 7.5 (or higher). However, it should be noted that each demo under default website must have a separate Application Pool. We must deploy the Go>Sign Applet Demo in a separate pool, else an error will be received. For deployment, follow the instructions below:

1. Open IIS Manager.
2. Right click on the **Application Pool** and add a new Pool by using a unique name and keeping other values as default.
3. Open **Sites** > **Default Web Site** from the navigation tree in the left-hand pane.
4. Right Click the **Default Web Site** and then click on **Add Application** to add demo as an application and select the application pool which you created.

The below points must be kept in mind while performing the above steps:

1. *The pool name and alias of demo application should be the same as demo name.*
2. *Each Go>Sign Applet Demo have their own copy of the available_demos.html which could be used to access the deployed demos.*

9 Updates

There are no new updates or changes in this release compared to the previous release.

*** End of document ***